NAS5-10667

ONBOARD PROCESSOR

ADDRESSING AND PACKAGING EVALUATION

FINAL REPORT

April 30, 1970

Prepared For

NASA GODDARD SPACE FLIGHT CENTER

Greenbelt, Maryland

By

WESTINGHOUSE DEFENSE AND SPACE CENTER

Aerospace and Electronics Systems Division

Baltimore, Maryland

W. Moffett Tharpe, Jr
572

Nas 5- 10667

ONBOARD PROCESSOR
ADDRESSING AND PACKAGING EVALUATION

FINAL REPORT

April 30, 1970

Prepared For

NASA GODDARD SPACE FLIGHT CENTER

Greenbelt, Maryland

By

WESTINGHOUSE DEFENSE AND SPACE CENTER

Aerospace and Electronics Systems Division

Baltimore, Maryland

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

LIST OF ILLUSTRATIONS

LIST OF TABLES

1.0 INTRODUCTION

This report summarizes the work performed on NASA Contract NAS5-10667 modification number three. Objectives of this contract were to improve the programming capability of the OBP system and at the same time reduce CPU control circuitry for improved packaging feasibility. The Statement of Work, associated with this contract, was divided into three main tasks. The first task, Task 1, dealt with the investigation of two techniques for improving the addressing capability of the OBP. The evaluation of Task 1 indicated that neither addressing approach was suitable for the OBP system and as a result a new approach was selected for Task 2. Section 3.0 of this report lists those changes selected for Task 2. After these changes were finalized, they were designed into the CPU, and drawings were updated. The changes associated with Task 2 were implemented so that the control circuitry of the CPU was reduced. This reduction in control circuitry aided the implementation of Task 3.

The objective of Task 3 was to evaluate the control circuitry on the basis of redesigning it for better partitioning. Existing control circuitry in the CPU was organized in groups that had similar logic structures. These groups were then used in the partitioning of the central logic. The results of this reorganization were evaluated and it was felt that a more efficient approach to partitioning could be taken. Therefore, a preliminary investigation into the use of read-only memories for the control section was undertaken since this approach appears to simplify the partitioning task. The results of this study are also presented in the report.

2.0 TASK 1

The purpose of Task 1 of this report was to evaluate two addressing techniques for the OBP-CPU. These two addressing approaches were chosen as the most desirable approaches to consider at the outset of the contract. Direct inputs from OBP programmers indicated both approaches would provide the desired addressing flexibility. Therefore the selection of the preferred approach was primarily dependent on its impact on the hardware. The following addressing techniques were evaluated.

2.0.1 Outline of Changes for Task 1

The following outline illustrates the changes to the CEU for Techniques I and II.

2.0.1.1 Technique I

Modifications:

1. Bits 7-12 replace bits 1-5 for minor op-code decoding.
2. Bits 1-6 of the instruction word are used as the immediate operand.

Instructions Added

1. LOAD A IMMEDIATE
2. PLUS A IMMEDIATE
3. MINUS A IMMEDIATE
4. LOAD EA IMMEDIATE
5. PLUS EA IMMEDIATE
6. MINUS EA IMMEDIATE

2.0.1.2 Technique II

Modifications:

1. Bits 12 and 13 of the instruction word are used as an addressing mode field for decoding one of four addressing modes.
2. Addition of immediate addressing to applicable major op-code instructions. An 11 bit immediate operand is used.

3. Addition of indirect addressing to applicable major op-code instructions. A 5 bit page register is required to maintain the 16 bit address.

2.1 Addressing Technique I

The discussion of this technique is divided into two sections. The first section discusses the general modifications required in the CPU and the second section discusses the specific details of the additional instructions.

2.1.1 Modifications to the CPU

The implementation of this approach, required a change to the instruction field decoding. A total of six instructions was added to the instruction set and all six were decoded as minor op-code instructions. Since these instructions added immediate addressing capabilities to the CPU, it was desirable to utilize the least significant six bits of the instruction word as the immediate value and this selection was incorporated into the design. This decision was made because of the available gating and flow organizations, and the inherent programming ease associated with the use of the least significant bits. The selection of the least significant six bits as the immediate value required a change in the minor op-code field decoding. In the original systems, bits 1-5 were decoded as the minor op-code. Due to the addition of the immediate instructions, the minor op-code field was moved to bits 7-12 and associated decoding was changed to satisfy decoding requirements.

2.1.2 Addition of Instructions

Technique I provided immediate addressing capability for the OBP

programs by adding six minor op-code instructions to the instruction set. These added instructions were similar to existing major op-code instructions with the difference being the use of bits 1-6 of the instruction word as an operand rather than part of an address for the operand fetch. The following instructions were investigated:

1. LOAD A IMMEDIATE
2. PLUS A IMMEDIATE
3. MINUS A IMMEDIATE
4. LOAD EA IMMEDIATE
5. PLUS EA IMMEDIATE
6. MINUS EA IMMEDIATE

2.1.2.1 Load Immediate Instructions

Two load immediate instructions, LOAD A IMMEDIATE and LOAD EA IMMEDIATE were evaluated for Technique I. These load immediate instructions gate the immediate operand field of the instruction word through the adder and into the selected register.

Two phases (ϕ_1, ϕ_2) are required to implement these instructions with phase one utilized as the normal instruction fetch phase. The second phase, ϕ_2, executes the immediate load by gating the immediate operandiinto the adder which in turn is clocked into the selected register. Since load functions do not require carry delays through the adder, only one clock cycle is necessary to implement the load. A total of three clock cycles are required to complete the load immediate instructions.

The existing instructions, LET and SET EXTENSION WITH, are similar in function to the load immediate instructions. In ϕ_2, the existing instructions

fetch an operand from memory and load the selected register. The fetch-load function is executed in three clock cycles, for a total of five clock cycles for complete execution. Since the load immediate instruction requires three clock cycles for completion, two clock cycles are saved.

The total hardware impact of these instructions was minimal, with the majority of the changes being implemented in the control area. The MORA, SUMACC, SUMMQ, MQ clock and ACC clock control circuits require modification. In addition, logic was added to decode these instructions and the adder inputs were changed. The gates controlling the most significant 12 bits of the A input to the adder were changed from two inputs to three inputs in order to inhibit these bits during the load immediate instructions.

2.1.2.2 Plus Immediate Instruction

Two plus immediate instructions, which add the immediate value to the accumulator or to the extended accumulator, were evaluated for Technique I. The plus immediate instructions gate the immediate value of the instruction word into one set of adder inputs and the selected register into the other set of adder inputs. Then, the adder output is clocked into the selected register.

These two instructions have similar phasing and are implemented by utilizing much of the same gating. These instructions require two phases (ϕ_1, ϕ_2) with phase one used as the normal instruction fetch phase. Phase two (which requires 2 clock cycles) gates in the immediate portion of the instruction word, and inhibits the remaining inputs to that set of adder inputs and also gates the selected register into the other set of adder inputs. The adder is then clocked into the selected register and the instruction is completed in four clock cycles. In the original CPU design, the PLUS instruction (which

is very similar to the plus immediate instructions) adds the MOR to the ACC. During the second phase of the instruction, the operand fetch and add are executed. These two operations are completed in three clock cycles. Thus, the PLUS instruction is five clock cycles in duration and only one cycle longer than the plus immediate instructions.

Very few additional gates were required to implement the plus immediate instructions since existing phasing was used to implement the two instructions. Circuitry was required for the decoding logic, MORA, ACCB, MQB, SUMACC, and SUMMQ control lines as well as the ACC and MQ clocks.

2.1.2.3 Minus Immediate Instructions

Two minus immediate instructions, which subtract the immediate value from the accumulator or from the extended accumulator, were evaluated for Technique I. The minus immediate instructions subtract the immediate portion of the instruction word from the selected register and then store the result in the selected register.

Two methods for implementing these two instructions were considered. The first approach investigated the possibility of utilizing a load immediate into the register not being selected and then negating this value. This approach destroyed the contents of the unused register and thus was discarded as unacceptable.

The second approach, which was selected for the design, evaluated the subtraction of the immediate value from the selected register.

The organization of the two minus immediate instructions is similar to the MINUS instruction requiring two phases (ϕ_1, ϕ_2) and utilizing phase one as the instruction fetch phase. Phase two which requires 2 clock cycles,

gates the complement value of the immediate operand to the least significant six bits of one set of adder inputs and one's into the remaining 12 inputs. Simultaneously, the selected register, along with the Co carry are gated into the other set of adder inputs. The result of this 2's complement addition is then clocked into the selected register. Two clock cycles are required to complete ϕ_2. A total of four clock cycles are required to complete the minus immediate instruction, as compared with five clock cycles for the original MINUS instruction.

Most of the additional logic for the minus immediate instructions was added in the control with only the adder inputs being changed in the arithmetic-register area. The 12 most significant gates of one set of adder inputs had one input gates changed to two input gates for use in "gating on" those 12 most significant inputs of the adder. The "gating on" feature set one's into these inputs so that a 2's complement addition could be performed. In the control, coding and phasing gates were needed to implement the two phase operation associated with these instructions. Additional gating was required for the $\overline{\text{MOR}}$ A, ACC B, CO, MQB, MQ clock and ACC clock signals. The change to the register-adder section amounted to changing the one input gates to two input gates on the 12 most significant bits of the adder so that 1's can be forced into these bits.

2.2 Addressing Technique II

Addressing Technique II did not add instructions as such to the OBP, but added two addressing capabilities (immediate and indirect addressing) which affected a number of instructions. Only modifications to the CPU were nesessary to implement these changes and they are discussed in the following paragraphs.

2.2.1 Design Modification

During the initial evaluation of this addressing approach, the removal of the index register was considered to facilitate implementation of this technique. After the initial investigation of the OBP program was completed, it was decided that a definite need for indexing existed, and the index register was not removed.

With these two addressing modes added to the CPU, a total of four addressing modes were available in the CPU and some type of detection circuitry was necessary. To properly detect these modes, it was decided that bits 12 and 13 of the instruction word would be used as a 2 bit field for address mode selection. The addressing modes and their associated codes are listed in Table 2-2.

Immediate addressing capability was added to applicable instructions as referenced in Table 2-1. This addressing feature permitted the use of the least significant 11 bits of the instruction word as an immediate operand. Since the basic flow organization of the CPU was such that operands were gated through the adder and the adder also controlled register to register flow, it was advantageous to use the existing flow organization for the immediate addressing scheme.

Because bit 12 of the instruction word was used for address mode detection, and an 11 bit immediate operand would satisfy the majority of computational requirements, an 11 bit immediate operand was selected for implemention in the CPU. The use of only an 11 bit operand required that the decoding of the immediate mode would control the 7 most significant inputs of the adder which were utilized when the full 18 bit operand was selected.

TABLE 2-1

Control Modifications For

Instructions Utilizing Immediate Addressing

Seven MSB of Adder Gated Off	Seven MSB of Adder Gated On	Six LSB of MOR Used Directly
Plus	IF SUB	SET SCALE
TRANSFORMED BY	IS EQUAL	SHIFTED BY
LET	IS GREATER	CYCLED BY
ANDED WITH	IS LESS	DOUBLE SHIFTED BY
TIMES	MINUS	DOUBLE CYCLED BY
THEN GO TO		
ORED WITH		
SET EXTENSION WITH		
USE SUB		
GO TO		
DIVIDED BY		
EORED WITH		
SUB PLUS		
EXECUTE		
HALT		
IO		
RESUME		

TABLE 2-2

Addressing Mode Selection

Address Mode Field		Addressing Mode
13	12	
0	0	DIRECT ADDRESSING
0	1	IMMEDIATE ADDRESSING
1	0	INDEXING
1	1	INDIRECT ADDRESSING

Two sets of adder inputs had to be controlled so that both "gating on" and "gating off" provisions were present. In order to accomplish this, the "A" inputs to the adder were changed as follows. First, the seven most significant two input gates were changed to three input gates for "gating off" control. Second, the seven most significant one input gates were changed to two input gates for "gating on" control. Also associated with the immediate addressing, was the inhibit control on the memory request lines, as well as associated clock and input control modifications. In order to properly utilize the immediate approach in shift, cycle and scale setting instructions, control circuitry was added to control the setting of the scale register and operation counter with the least significant six bits of the instruction word. Table 2-1 lists the instructions and their associated control.

In general, the execution of instructions using the adder was decreased by only one clock cycle with immediate addressing. In the original design, the fetch and add phase were overlapped so they could be executed in three clock cycles. To decrease this time either a faster adder must be utilized or the memory cycle time for the OBP must be decreased.

Indirect addressing was implemented for the instructions referenced in table 2-1. Indirect addressing uses the normal operand fetch phase as an address fetch phase which fetches the address used to fetch the operand. This additional phase adds two clock cycles and an extra memory cycle to each instruction utilizing this mode of addressing.

The original system was designed to access 65K of memory through a 12 bit address field and a 4 bit page which was appended to the address field. The Page Register value was loaded into the four most significant bits of the

address register during the instruction fetch phase. Thus, the first 4096 core locations could be accessed without setting the page register. When higher core locations were accessed, the Page Register had to be set with a Set Page instruction prior to executing the given instruction.

When bit 12 of the instruction word was selected as a field bit for address mode selection, only an 11 bit address field was available to address memory. The use of an 11 bit address field only allows access of the first 2048 locations of core, therefore some means of controlling the 12th bit of the address field was required. Since the present CPU organization utilized the Page Register for address control, the utilization of this organization would enable complete address modification with minimal impact on hardware. Therefore, an additional bit was appended to the Page Register.

With the introduction of indirect addressing, an additional control phase similar in structure to the index phase was added to the CPU. The decoded output of the address mode selection field selected this phase to fetch an address, which was clocked into the address register and used as the normal operand address. The associated control circuitry was modified when this phase was added to the CPU and additional control was added to request memory during the indirect address fetch. Since the instruction word bits 12 and 13 were decoded for the address mode selection, it was also necessary to ensure that the gate delays through the decoder to the phase selection circuitry was not critical to the system operation. Therefore, the number of gates connected in a serial fashion had to be limited. This necessitated the use of a parallel gating structure and resultediin a greater number of gates.

2.3 Summary of Task 1

Task 1 was organized so that two addressing techniques were evaluated for the OBP system. A summary of these techniques follows:

2.3.1 Technique I

The addition of immediate addressing instructions was evaluated in Technique I of this report. Six instructions were added to the instruction set providing the capability to use immediate operands in OBP programs. The following six instructions were added:

1. Load A Immediate
2. Plus A Immediate
3. Minus A Immediate
4. Load EA Immediate
5. Plus EA Immediate
6. Minus EA Immediate

The addition of these instructions required the change of the minor op-code field from bits 1-5 to bits 7-12 of the instruction word since bits 1-6 are used as the immediate operand field. It was possible to use bits 7-12 as the immediate operand, but that approach had a significant impact on hardware beside the confusion associated with using these bits as an immediate value.

The purpose of adding immediate instructions to the CPU is to execute instructions utilizing partial word operands with a decrease in execution time. The savings in program time however, must bejjustified by a limited increase in hardware. While the total impact on hardware associated with these instructions was minimal, the savings in program time was very

limited. The immediate add, or immediate 2's complement add (subtract) instructions are only one clock cycle faster than the original PLUS and MINUS instructions and this minimal saving in execution time does not justify the implementation of this technique. In addition, the OBP programmer expressed their feeling that more addressing capability than that gained with this technique was desired for increased programming flexibility. Thus, the technique was rejected.

2.3.2 Technique II

The addition of immediate addressing and indirect addressing capabilities for the OBP system were evaluated in Technique II of this report. The implementation of this technique dictated that some type of addressing mode selection scheme must be implemented in the CPU to distinguish between the four possible addressing modes. As a result, bits 12 and 13 of the instruction word were selected as the field for use in address mode selection. These bits were decoded and are referenced in Table 2-2.

Since the immediate addressing technique utilized the least significant 11 bits of the instruction word as an immediate operand. The need for an operand fetch was eliminated. Because the immediate operand did not use all 18 bits, a method of controlling the remaining seven bits was required. Consequently, the adder gating was changed. Since the operand fetch and add phases were overlapped in the original CPU instruction, a gain of only one clock cycle was realized for any immediate instruction fully utilizing the adder. While the additional circuitry associated with the immediate addressing approach was minimal, it was not justified by the limited savings in execution time.

When indirect addressing was incorporated into the CPU, the addition of a bit to the Page Register, the control of the memory request line, and

the new phasing added, had a big impact on the control circuitry of the CPU. In addition, indirect addressing adds two clock cycles to the execution time of applicable instructions. The additional execution time and associated increase in hardware were the main factors in rejecting this approach.

It was desirable to reduce execution time in the approaches discussed above so that program efficiency would be improved. Two means of reducing instruction execution time were considered; either design a faster adder, or reduce the basic memory cycle time. Since these two changes would have a considerable impact on hardware, they were rejected.

After evaluating both techniques, it was decided that the goals which originally motivated these changes (increased program execution speed and flexibility with minimal hardware impact) were not achieved.

3.0 TASK 2

The study associated with Task 1 dealt with the investigation of two approaches for improving the addressing capability of the OBP system. These two techniques were evaluated and then discussed with NASA personnel and programmers associated with the OBP. The results of these discussions led to the decision to eliminate both addressing approaches since neither proved satisfactory for the OBP system. Instead, two alternative approaches for improving the addressing capability of the OBP were suggested.

First, multiple index registers could be added to the CPU. Second,

the one index register could be utilized along with indirect store and indirect load instructions. The second approach was chosen primarily because of the increased hardware associated with the first approach.

In addition to the two indirect instructions, it was requested that CPU registers be available for use as loop counters for controlling both program loop execution and branching. To satisfy this request, instructions to test and increment registers were added to further increase the addressing and operational features of the OBP system.

Accompanying the test instructions was the request for the ability to manipulate and transfer data in the CPU registers. In particular, it was felt useful to compute an index value in the accumulator and transfer this value to the subscript register (SS), while not destroying the present value of the SS. This operation is performed in the present OBP by executing the following four instructions: PLUS, SAVE SUBSCRIPTS IN, YIELD, and USE SUBSCRIPT. These four instructions could be replaced by PLUS and EXCHANGE instructions. In order to satisfy these requests, three instructions were added to the instruction set to exchange the accumulator,the extended accumulator, and the subscript register.

Although the new instructions were proposed with the improvement of system performance in mind, the decision to implement them was subject to consideration of the impact of integrating these instructions into the design. It was particularly desirable to delete any circuitry not being used in the present system applications. The investigation of the hardware impact of these instructions was carried out with this thought in mind and it was found that certain instructions and functions were not utilized by the programmers in the

present programs. As the evaluation of implementing Task 3 was concluded, the instructions were added with the least possible amount of additional hardware, and all circuitry having limited application in present programs was removed. The following changes were incorporated.

3.0.1 Additional Instructions

1. LOAD INDIRECT
2. STORE INDIRECT
3. IF EA $\neq$ 0, SET D[4] and INCREMENT EA[1]
4. IF SS $\neq$ 0, SET D and INCREMENT SS[2]
5. EXCHANGE A[3] & SS
6. EXCHANGE A & EA
7. EXCHANGE EA & SS

NOTE: (1) EA - Extended Accumulator

(2) SS - Subscript Register

(3) A - Accumulator

(4) D - Decision Flip Flop

(5) MOR- Memory Operand Register

3.0.2 Functions Deleted

1. SCALE REGISTER
2. OR-AND FLIP FLOP

3.0.3 Instructions Removed

1. LET SCALE
2. SET SCALE
3. OR
4. AND

3.0.4 Instructions Modified

1. NORMALIZE - The normalized count is now stored in the subscript register
2. MULTIPLY - Scaling is deleted from multiply
3. DIVIDE - Scaling, correction cycles, and overflow tests are deleted from divide.

The chart in Table 3-1 illustrates the impact of these changes.

3.1 Addition of Instructions

Based on the results of Task 1, seven instructions (specified in Paragraph 3.0.1 of this report) were added to the OBP instruction set. The intent of these instructions was to improve the programmers ability to write programs directly associated with the OAO satellite applications.

The organization of the CPU is such that data flow paths interconnect the register circuitry via the adder. These flow paths were used to implement the new instructions and thus only the circuitry associated with the CPU control logic was increased.

3.1.1 Instruction Description and Discussion

Seven new instructions have been selected for integration into the CPU design. A discussion of these instructions is given in the following paragraphs.

3.1.1.1 Indirect Instructions

Two indirect instructions, INDIRECT LET and INDIRECT YIELD were selected to be incorporated into the CPU design. The selection of these two instructions was based on the desire for additional capability to easily access buffer memory areas not readily accessible with one index register. This

CHIP TYPE/ (# Gates/Chip)	9046/(4)	9047/(3)	9044/(2)	501/(4)	9042/(2)	933/(2)	9040/(1)	Total Gates Deleted
DELETIONS								
SCALE REG	12	8	1	30				51
LET SCALE	3	1		1	1			6
SET SCALE	1	3						4
MULTIPLY	17	10	4	2	1	1	1	22
NORMALIZE	1							1
OA F/F	7	5	3	4	6		1	26
OR	1	1		1				3
AND	1	1		1				3
DIVIDE (Total)	31	18	11	5		1	15	81
DIVIDE (Scaling)	(7)	(3)	(2)	(2)			(1)	(15)
Total Gates/Chip Type	74	47	19	44	8	2	17	211
ADDITIONS								Total Gates Added
INDIRECT LOAD	6	4		1				11
INDIRECT STORE	3	4						7
EXCHANGE A&EA	5	2						7
EXCHANGE A&SS	5	2						7
EXCHANGE EA&SS	5	1	1					7
IF EA≠0,SET D&increment	3	3						6
IF SS≠0,SET D&increment	4	2						6
Total Gates/Chip Type	31	18	1	1				51

Total Gates Removed = (211-51) = 160

Total Integrated Circuits Removed = 60

Table 3-1. Gate Changes for Task 2

increased capability could be achieved with multiple index registers. However, indirect addressing provided similar capabilities with less hardware.

The indirect load instruction loads data in the accumulator in the same manner as the LET instruction, except that the indirect load requires an additional phase which is two clock cycles long. This phase fetches an address which is set into the address register, which in turn is used to fetch the normal operand fetch address. Since all memory access functions require two clock cycles, the indirect load is two cycles longer than a LET instruction. Three phases (ϕ_1, ϕ_2, ϕ_3) are required to implement this instruction. Phase one is the normal instruction fetch phase, phase two fetches the address for the operand fetch, and phase three fetches the operand and loads it into the accumulator.

The indirect store instruction stores data in memory in the same manner as a YIELD instruction except for the data storage phase. The indirect store instruction requires an additionalpphase used to fetch an address that is set into the address register and used as an address to store data. The indirect store requires three phases (ϕ_1, ϕ_2, ϕ_3) for complete execution and is one clock cycle longer than a YIELD. Phase one is the normal instruction fetch phase, phase two fetches the address used for data storage, and phase three stores the data at that address.

Both instructions were decoded as major op-code instructions with indexing available for both. The associated phasing, decoding, and control modifications were minimal due to the use of existing phasing and flow organizations. Control was also added to the memory request logic for the extra memory cycles. Eighteen (18) gates were required to implement these instructions.

3.1.1.2 Test for Zero Instructions

Two instructions (TEST SS and TEST EA) were added to the instruction set to test register contents for zero. These instructions required no operand fetch and thus were implemented as minor op-code instructions. They permit the SS and EA registers to be used as pointers or counters for calling up sequential arrays of data, and for performing loop and branch operations. The instructions were implemented in two phases (ϕ_1, ϕ_2) and are completed in five clock cycles. The instructions operate as follows.

Phase one executes the normal instruction fetch function, while phase two performs the test for zero on the selected register. If the register under test is zero, no action is initiated in phase two and the instruction is completed, but if the register tested in phase two is not zero, the "D" flip flop is set and the register under test is incremented by one.

The implementation of these instructions required little additional hardware since existing zero detection circuitry was used for performing the tests. Each register under test is gated into the adder and the "SUM=0" output is tested for a "1". This output is then used to control the setting of the "D" flip flop and the incrementing of the register. No new circuitry was introduced into the registerarithmetic section of the CPU and only a small amount was added in the control area. Full use of existing flow paths was employed to hold the additional logic to a minimum. Twelve (12) gates were required to implement these two instructions.

3.1.1.3 Exchange Instructions

Three of the seven new instructions are "register exchange" instruc-

tions. These instructions consist of exchanging the accumulator and the extended accumulator, exchanging the accumulator and the subscript register, and exchanging the extended accumulator and the subscript register.

Addition of these instructions allows the programmer to exchange registers without using several memory access instructions. One particular application of the exchange accumulator and subscript register is in the computation and use of an index value. First the index value is computed in the accumulator, then the accumulator and subscript registers are exchanged so that the computed index values can be used immediately without destroying the existing index value.

The exchange instructions consist of two phases (ϕ_1, ϕ_2), and are five clock cycles in duration. No operand fetch is required with these instructions and they are decoded as minor op-code instructions. Phase one is the normal instruction fetch phase and phase two performs the exchange. When executing an exchange with the A, the following sequence is executed in phase two. The MOR is cleared and the A is gated through the adder and stored in the MOR. Then the EA or SS (depending on the instruction being executed) is gated through the adder and is clocked into the A. The MOR is then gated through the adder to the EA or SS and the register is clocked.

If the instruction exchanges the EA and SS, the following sequence is executed. During phase two, the MOR is cleared and the EA is gated through the adder and stored in the MOR. Then the SS is gated through the adder and clocked into the EA. Finally, the MOR is gated into the adder and clocked into the SS to complete the instruction.

These instructions make use of the existing register organization, with register to register flow accomplished via the adder and therefore only

control circuitry modifications are required. Twenty-one (21) gates are required to implement these instructions.

3.2 Deletions - Circuitry and Instructions

In addition to the seven instructions added in Task 2, the following circuitry and instructions were deleted.

The following deletions were made:

1. Removal of the OA flip flop.
2. Removal of the scale register.

The following instructions were modified:

1. Normalize
2. Multiply
3. Divide

The following instructions were removed:

1. LET SCALE
2. SET SCALE
3. OR
4. AND

The decision to remove registers and instructions was a combined hardware-software decision. When the two indirect instructions were added, two major op-codes were required and only one major op-code decoding was available for these instructions. The two alternatives which existed were either to add additional circuitry for the decoding, or to delete existing major op-code instructions. To properly evaluate the alternatives, the hardware and programming aspects of these changes were investigated. An evaluation of the first alternative, the addition of decoding circuitry, verified there would be a considerable impact

on the hardware and programming manual documentation and this alternative was rejected. The second alternative was to investigate the possibility of removing major op-code instructions. An evaluation of this alternative showed that removal of the seldom used scaling function would free a major op-code. The following section is a discussion of the removal of this function.

3.2.1 Scale Register and Associated Instructions

In addition to its limited use, the scale register and its associated circuitry were non symmetrical relative to other register and control logic. This made it difficult to partition the scale register logic and since a total of 113 gates and flip flops were associated with the scale register, its removal was attractive from the standpoint of hardware simplification.

The deletion of the scale register eliminated the LET SCALE instruction (major op-code) and the SET SCALE instruction (minor op-code). In addition, three other instructions NORMALIZE, TIMES, and DIVIDED BY were affected by the removal of the scale register. In all, a total of 113 gates and flip flops were removed by the elimination of the scale register.

First, the NORMALIZE instruction, which previously stored the normalized count in the scale register, had to be changed. Since the normalized count must be saved, two possible solutions were considered. First, the count could be stored in a fixed memory location. Second, the count could be stored in a CPU register. The register store approach was considered more practical since extra logic and execution time is required for the memory store-cycle of the first approach. Storage of the normalize count in a register utilizes the basic flow paths of CPU, and requires negligible change to the logic. An additional six gates are required on the adder. However, these

replace the six originally used on the scale register. The subscript register was selected as the register to be used for storing the normalized count since the accumulator and extended accumulator are normalized during the NORMALIZE instruction. Only a small amount of control logic is required to implement this transfer.

A second instruction affected by the removal of the scale register was multiply (TIMES). TIMES originally had a scaling phase that shifted the product, based on the value in the scale register. The scaling phase and its associated hardware were eliminated with the removal of the scale register.

The third instruction affected by the scale register was DIVIDED BY. The divide instruction had a scaling phase that shifted the dividend prior to performing the divide itself. Again, the direction and amountoef the shift was determined from the value in the scale register. The scaling phase and its associated hardware were removed with the elimination of the scale register.

3.2.2 Modification of the Divide Instruction

In the original design of the CPU, the decision to implement a total hardware divide was based on the evaluation of the usage of a divide instruction. The initial investigations indicated that the divide would be used frequently and would prove most efficient, both in power saved, and execution time saved, if it was totally performed with hardware. Further investigations into the application of the divide instruction in the system programs indicated that the divide instruction was used to a very limited extent. This knowledge, accompanied with the fact that divide used a considerable amount of control logic, prompted the decision to eliminate the majority of the circuitry associated with divide.

The original hardware divide made both the divisor and dividend positive prior to performing the divide algorithm so that no correction to the quotient would be required. After the divide was completed, the sign of the remainder was tested to see if it was negative. If it was negative, the divisor was added to the remainder for the correction cycle. After the correction cycle was completed, the quotient was placed in the accumulator and the remainder in the extended accumulator. Also included in the hardware divide were the divide overflow tests.

It was decided that all special phasing associated with the divide algorithm would be removed and the divide instruction was reduced to perform the standard add/subtract shift cycles of the non-restoring division algorithm with 81 gates and flip flops eliminated by the modification. As a result, prior to the divide, the programmer must ensure that the divisor is larger, in magnitude than the dividend to prevent overflow from occuring. In addition, if an exact quotient (quotient may be off by 1 in the least significant bit) or remainder is desired the appropriate corrections cycles must be performed as indicated in Table 3-2.

3.2.3 Elimination of the OR-AND Flip Flop

The OR-AND (OA) flip flop was initially designed into the CPU as a means for controlling the setting and resetting of the "D" flip flop. The setting of the OA indicated an "and conditional test" to reset the "D" flip flop, while the resetting of the OA indicated an "or conditional test" to set the "D" flip flop. An examination of the system programs indicated that only the "OR" state of the OA flip flop was being used. Since only one state of the flip flop was being used, the decision was made to eliminate it.

CONDITIONS				CORRECTIONS	
Sign of Dividend	Sign of Divisor	Sign of Residue After Division	Value of Residue After Division	Quotient Correction	Residue Correction
Positive	Positive	Positive	≥ 0	No correction	No correction
Positive	Positive	Negative	< 0	No correction	Add divisor
Negative	Positive	Positive	> 0	Add low-order one	Subtract divisor
Negative	Positive	Positive	$= 0$	No correction	No correction
Negative	Positive	Negative	$= -$ Divisor	No correction	Add divisor
Negative	Positive	Negative	< 0 and $\neq -$ Divisor	Add low-order one	No correction
Positive	Negative and $\neq -$ Dividend	Positive	≥ 0	Add low-order one	No correction
Positive	Negative and $= -$ Dividend	Always = - Divisor		No correction	Add divisor
Positive	Negative	Negative	< 0	Add low-order one	Subtract divisor
Negative	Negative	Positive	> 0	No correction	Add divisor
Negative	Negative	Positive	$= 0$	Add low-order one	No correction
Negative	Negative	Negative	$=$ Divisor	Add low-order one	Subtract divisor
Negative	Negative	Negative	< 0 and $\neq$ Divisor	No correction	No correction

Table 3-2. Division Corrections

Directly associated with the elimination of the OA flip flop was the deletion of two minor op-code instructions; the OR and AND instruction. Removal of these two instructions, the OA flip flop and the setting of the "D" flip flop resulted in a decrease in control circuitry.

3.3 Summary of Task 2

Task 2 was organized to make changes in the CPU design that would give the programmer greater flexibility in addressing and other programming operations. The total impact on hardware was the prime factor in deciding which approach to take.

A total of seven new instructions were added to the OBP instruction set. Table 3-1 illustrates that a total of 51 gates were required to fully implement these seven instructions. The additional flexibility and programming speed gained from these instructions was large, relative to the percentage increase in hardware to implement the seven instructions. The hardware was held to a minimum by utilizing the existing CPU flow paths to implement these instructions and the circuitry added was entirely in the control area.

Four instructions, the scale register and the OA flip flop were removed. TIMES, DIVIDED BY, and the NORMALIZE instructions were affected by the removal of the scale register. In addition, DIVIDED BY had overflow tests and setup and correction cycles removed from the hardware.

The total change in the circuitry resulted in a reduction of circuits used in the CPU. While the seven instructions added circuitry to the control area, the deletions removed much more control circuitry. In comparing the additions with the deletions, the additions tend to follow the basic symmetry of the CPU design, while the deletions tended to be unsymmetrical. A total

decrease of 160 gates was realized with the improved instruction set. Table 3-1 indicates where the 160 gates were removed from the logic.

The elimination of unique type control logic from the CPU was most desirable from a packaging standpoint. The basic register-arithmetic portion of the CPU is readily partitioned, while the control logic does not appear to be readily partitionable. To enhance packaging efficiency of the control logic, the non-similar logic is to be minimized. The removal of the scale register and OA flip flop and simplification of the divide instruction eliminated a portion of that logic which is difficult to partition and thus should simplify the packaging of the control logic.

4.0 TASK 3

This section discusses the results of a study of methods for redesigning the OBP control logic in such a way as to introduce a greater degree of symmetry or regularity into its structure. It is hoped that such a redesign will facilitate partitioning the logic into a relatively small number of general purpose logic structures and thereby make it feasible to mount it on hybrid substrates.

In the breadboard, almost all inputs and outputs from individual gates require external connections by way of pins on the printed circuit boards. Since the number of pins is restricted, the density of gates is severely limited. It is hoped that by partitioning the logic into more complex structures with many of the interconnections made on the substrate itself, the gate density may be increased and the size of the processor reduced. To make this packaging method economically feasible, however, it is necessary that the number-of-distinct types of substrates be minimized. Thus complex logic

structures which occur repeatedly throughout the control circuitry must be identified.

4.1 Problems Encountered in Partitioning Control Logic

Unlike computer register logic in which essentially identical logic structures are associated with each bit, control logic is in general a formless irregular conglomeration of gates and flip flops. This is a result of the fact that this logic is used to generate all the unrelated timing and data dependent conditions which control the various data transfers and transformations within the computer. Each block of logic performs a function which is different from, and independent of, the functions of neighboring blocks.

A further complication results from the tendency of the logic structures which generate individual control signals to be relatively small groups of gates with large numbers of inputs and outputs. These external connections are difficult to reduce because they come from and go to a wide variety of places in the processor. For example, the signal which sets the END flip flop has over 20 inputs from such sources as the phase flip flops, various tests of the contents of the operation counter, selected bits of the memory operand register, instruction decoder, etc. It is impossible to package all these control signal sources on the same substrate with the END flip flop. As long as conventional gates and flip flops are used as the basic control building blocks, it appears that pin limitations are still the factor which control the degree to which packaging density may be increased.

4.2 Evaluation of Present Control Logic

Before a redesign of the OBP control logic was attempted, the

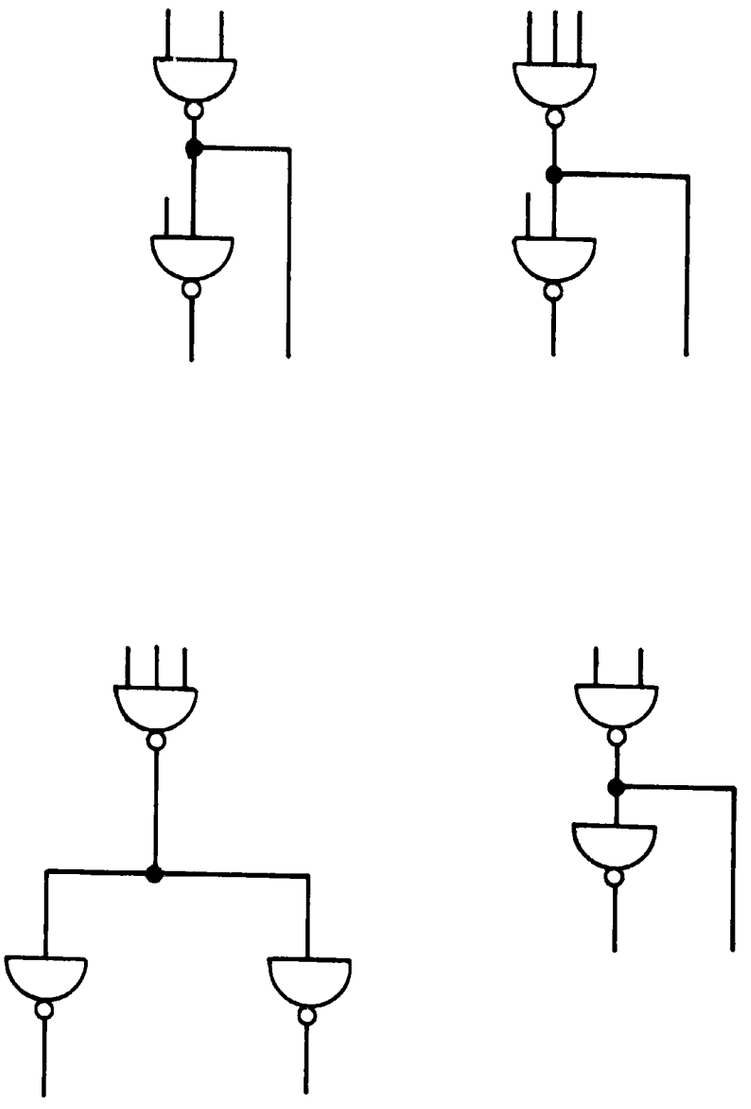
feasibility of partitioning the present design was evaluated. The logic was partitioned into as few types of fairly complex structures as possible. This not only provided an estimate of the magnitude of the problem but also served as a reference by which other designs could be judged. During a previous study, the register logic was partitioned into 18 sixty pin substrates of three different types. Since there are roughly the same number of gates in the control logic as in the register logic, this number provides an order of magnitude goal for the control logic partitioning.

As expected, the frequency of occurrence of similar logic structures sharply decreased as their complexity increased. In fact, no groups containing more than three gates were found which occurred often enough (e.g., more than eight times) to be considered general purpose structures. Examples of the recurring structures which were found are shown in Figure 4-1. It can be seen that the savings in external connections is small as long as the occurrence of similar complex structures is so limited.

Using these small general purpose blocks and implementing the rest of the control logic with discrete gates and flip flops, a partitioning scheme was organized which required 60 substrates (assuming 60 pins per substrate) of five different types.

4.3 Approaches to Control Logic Redesign

Two approaches to the redesign of the OBP control logic were investigated. The first consisted of constructing several relatively complex logic structures which could serve as general purpose logic blocks. These blocks were not identical to any block actually appearing in the control, but were similar to several slightly different ones. Each block was designed to be substituted for a maximum number of similar structures with a minimum



S70-580-VA-5

Figure 4-1. Examp es of Recurring Structures

wastage of pins and gates. By replacing most of the control logic with a few groups of gates in this manner, a much greater degree of regularity was introduced without changing the basic building blocks of the system.

The other approach which appeared to offer a significant reduction in the complexity of the OBP control was the use of a microprogrammed LSI memory to store a majority of the control signals for the CPU. By sequencing through a block of locations whose outputs directly or indirectly generate the OBP control signals, many of the functions presently performed by discrete gates can be performed instead by a much more compact LSI memory.

In the following paragraphs, these two design approaches will be discussed in more detail.

4.3.1 General Purpose Logic Structure Approach

The functions performed by the control logic are determined by the various algorithms used to implement the instruction set. Given these constraints, however, the actual configuration of gates and flip flops necessary to perform the operations in the proper order is not fixed. In the original OBP control, the logic was designed with economy of gates in mind. However, symmetry is a more important characteristic for partitioning. Therefore, extra pins and gates can be included if by doing so it becomes possible to design one circuit which can replace each of several slightly different ones. The pins wasted by this practice will hopefully be more than compensated for by the pins saved through the use of a more complex interconnection pattern on the substrates.

To design these structures, the OBP control circuits were categorized by structural similarity; Then one general purpose circuit was designed to implement the switching function represented by each of these similar logic

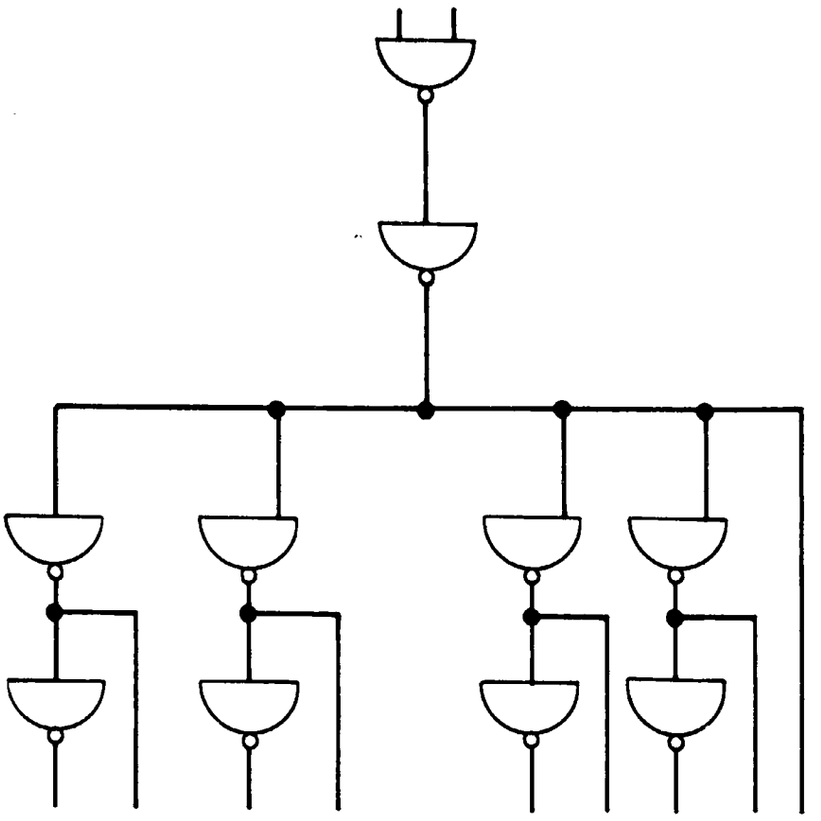
structures. In general this necessitated wasting inputs and even entire gates when the number of inputs varied from structure to structure. Output pins were wasted when some circuits required the true signal to be brought out, others required the complement, and some required that both be available. By providing the smallest number of external connections necessary to meet the needs of all the structures in the group, this waste was minimized.

Approximately 90 percent of the OBP control logic was implemented with 13 general purpose logic structures. Examples of these and their characteristics are shown in Figures 4-2 to 4-5 and Table 4-1. The 10 percent of the control logic which is not represented by these circuits consists primarily of miscellaneous single gates and expander inputs which can probably be mounted on the same substrates as the larger structures, thereby obviating the need for a special substrate type.

4.3.2 Partitioning for Substrates

Given the first order partitioning of organizing discrete gates into general purpose logic structures, there remains the second order partitioning problem of placing these structures on substrates. The logic structures were assigned to the substrates systematically to minimize both the total number of substrates and the wastage of gates. However, some wastage was unavoidable where logic structures would not fit on substrates in exactly the quantities required by the system.

In the partitioning of the control in its present form, it was assumed that the substrates to be used would have 60 pins. This assumption was made because all the hybrid substrates manufactured by Westinghouse thus far have been of this type. However, since control logic partitioning is so severely affected by pin limitations, it seemed reasonable to investigate the consequences of using substrates with more pins. A somewhat arbitrary decision



S70-580-VA-2

Figure 4-2. General Purpose Logic Structures

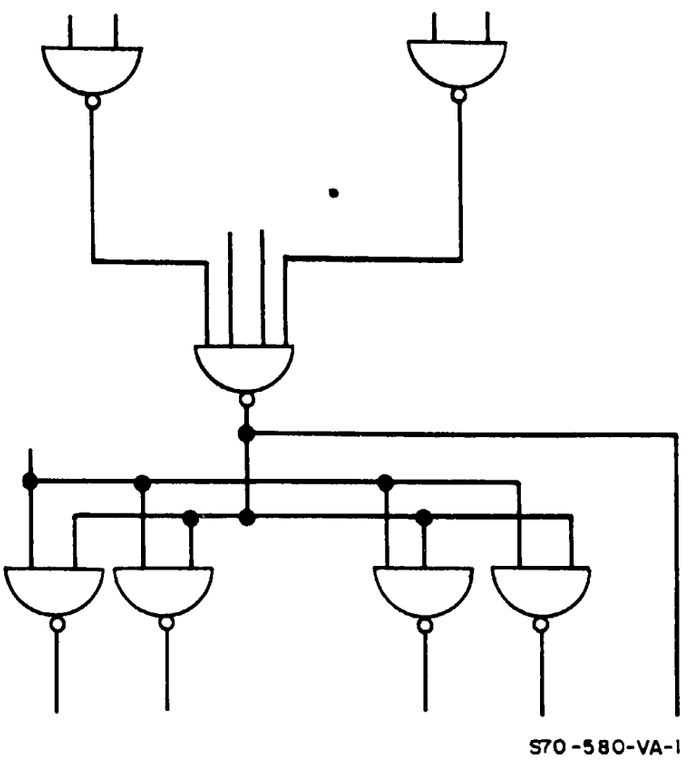
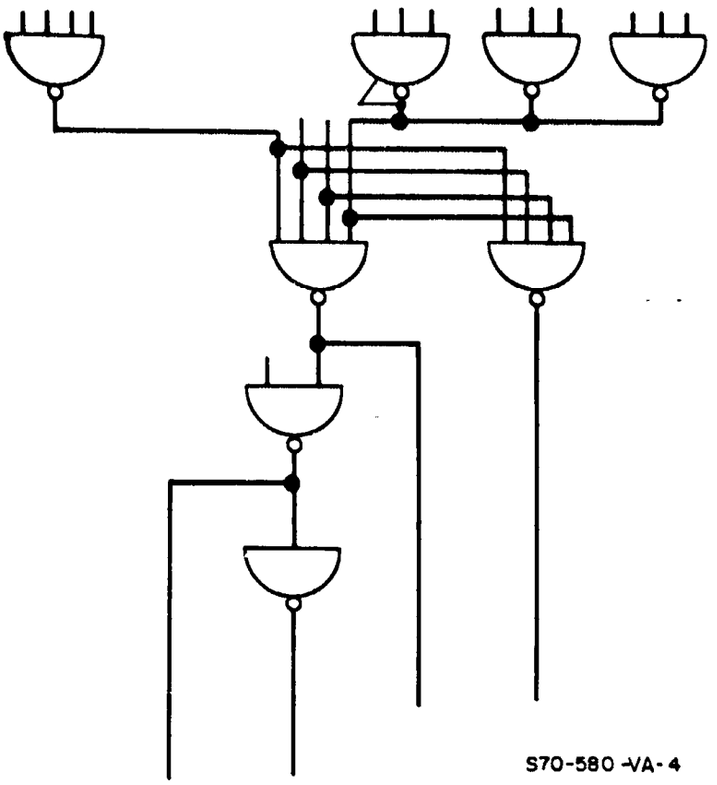
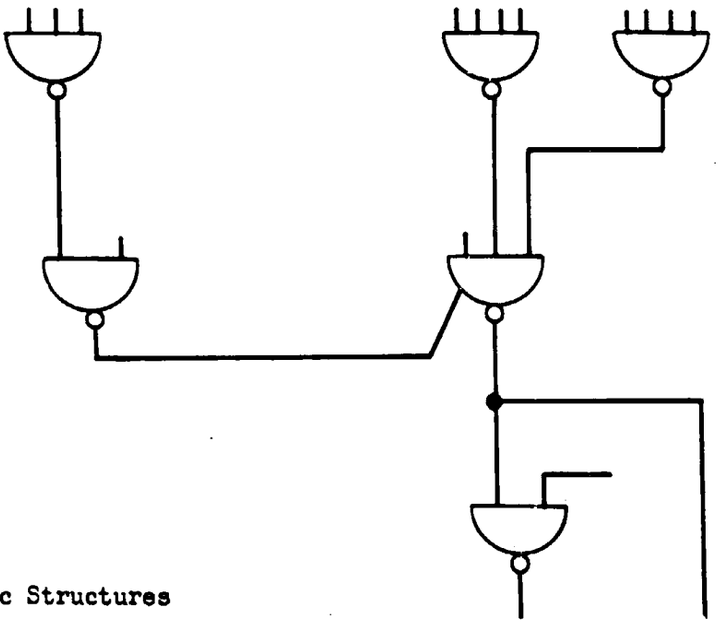


Figure 4-3. General Purpose Logic Structures



S70-580-VA-4

Figure 4-4



S70-580-VA-3

Figure 4-5

General Purpose Logic Structures

Logic Structure Type	No. of Gates	No. of Pins	Quantity Required
I	22	53	3
II	8	19	20
III	6	18	7
IV	5	9	7
V	14	28	8
VI	7	11	10
VII	6	16	9
VIII	3	7	41
IX	5	14	5
X	4	12	16
XI	10	19	8
XII	6	12	8
XIII	4	6	32

Characteristics of General Purpose Logic Structures

Table 4—1

was made to perform the partitioning using substrates having 60, 90, 120 and 150 pins. Increments of 30 pins were felt to be large enough to show significant differences in the partitions. The upper limit of 150 pins was chosen as the greatest number likely to be available on a substrate.

The results of this partitioning are summarized in Table 4-2. It can be seen that even if general purpose logic structures are available, the gains are small for 60 pin substrates. A large improvement is achieved by using 90 pin substrates. As the number of pins is increased to 120 and 150, further reductions in both the number of substrates and the number of types can be made. However, as the number of integrated circuit chips increases, the amount of substrate area used for interconnections also increases. Consequently, it may prove necessary to use larger substrates to physically realize the 120 and 150 pin configurations.

4.3.3 Summary of Control Logic Redesign

It appears that significant reductions in the total number of substrates and the number of substrate types may be achieved in this manner but only by relaxing the pin limitations to at least 90 pins per substrate. At present only 60 pin substrates are available but no fundamental reasons are known why larger numbers of pins and perhaps larger substrates (should overcrowding prove to be a problem) cannot be made available. It is also interesting to note (see Table 4-2) that the total number of external connections tends to be reduced by using substrates with more pins. This implies that an increase in reliability is possible through the use of general purpose logic structures since the primary sources of failures are wired connections.

Pins/Substrate	No. of Types	No. of Substrates	Max. No. of Chips/Substrate	No. of Pins (total)
60	7	52	13	3120
90	5	27	18	2450
120	4	22	22	2430
150	3	16	29	2400

Results of Logic Partitioning

Table 4-2

The substrate count for the control logic only approached the 18 substrate goal set by the register logic partitioning as the number of pins was increased to 120. The fact that such reductions cannot be achieved with 60 pin substrates demonstrates the problems caused by the inherently large numbers of external connections required for control logic.

4.4 Microprogrammed Control Logic

To design a microprogrammed control unit, each machine instruction is subdivided into "microinstructions", each of which represents the control operations needed during one period of the system clock. Each microinstruction is represented by one word in the memory. When a microinstruction is accessed, the bits of this word are used to provide the necessary control signals. Blocks of microinstructions are addressed in sequence to execute complex machine instructions. If data dependent conditions are required, the control memory output is used as an input to an external logic circuit which implements the necessary function.

In addition to data conditioning logic, circuits external to the memory will be needed to control the order and timing of the microinstruction addressing sequence. It has been estimated that the external logic will require approximately 300 gates or roughly 30 percent of the present control.

4.4.1 Partitioning the Memory

For the portion of control logic represented by the memory, partitioning is simple and efficient. First, since the interconnections other than address inputs and control signal outputs are all made on memory chips, difficulties due to pin limitations are greatly lessened. Secondly, since all the memory chips are connected in the same manner, only one type of

substrate need be designed for the entire memory. If read/write memories are used, all memory substrates will be exactly identical. If read-only memories are used, everything except the contents of the individual chips will be identical for all substrates. In either case, the use of one substrate type to implement 70 percent of the control logic represents a major achievement.

A preliminary design of the control memory needed for the OBP indicates that a memory of approximately 150 words by 120 bits should be adequate. This size assumes that all words are of uniform length and is therefore a maximum figure. Since all microinstructions do not make use of all the control outputs, it may be possible to design a memory with individual blocks of words containing only the control bits needed to perform particular microinstructions. For the following results, however, a uniform word length memory was assumed.

To permit an estimate to be made of the number of substrates required, it was further assumed that a 256 word by 8 bit memory chip was used as the basic building block. This is the size of one of the largest monolithic MOS memory chips currently available and was considered a realistic candidate for selection should this design approach be selected.

Using these assumptions it was determined that 15 such chips would be required. It was found that these could be placed on four identical 60 pin substrates.

4.4.2 External Logic Partitioning

Although a detailed partitioning scheme was not worked out for the control logic which was not incorporated into the memory, extrapolation from the results

obtained from the design of the general purpose logic structures described earlier should provide reasonably accurate estimates of what can be achieved. Assuming a linear relationship between the number gates and the number of substrates required, it should be possible to place the 300 external gates on 16 sixty pin substrates or six 90 pin substrates. If 90 pin substrates are used, it appears that all of the control logic could be mounted on approximately 10 substrates.

4.4.3 Problems Introduced by Microprogramming

One of the most serious drawbacks to the microprogramming approach described so far is the slow speed of presently available low power P-channel MOS memories. These memories typically have access times of the order of one or two microseconds. The clock presently used in the OBP has a pulse duration of approximately 320 nanoseconds and a period of 1500 nanoseconds. Since all operations must be completed before the clock pulse goes high, there are only 1180 nanoseconds per clock period in which to manipulate or transform data. It is obvious that problems occur with a one microsecond ROM in these circumstances since the time remaining after the control signals are available is insufficient for most processor operations.

One solution to this problem is to use a faster memory such as the available high speed bipolar TTL memories. However, the power dissipation for a bipolar memory of the required size would be more than 100 watts. This figure is clearly out of the range of interest for the OBP. On the other hand, CMOS memories are proposed for the near future which should operate at sufficiently high speed and require an extremely low amount of power. These may provide the best solution when they become available.

A second possibility is to reduce the clock frequency and thereby increase the period. It has been calculated that a reduction from the present 667 KHz to 400 KHz would allow sufficient time for a PMOS ROM to operate properly. It is beyond the scope of this report to determine whether a decrease in frequency of this magnitude can be tolerated. 400 KHz was shown to allow sufficient time to access a control word and perform an 18 bit addition in one clock cycle.

The third method is to compensate for the access time of presently available ROM's by overlapping the microinstruction fetch with the previous control word. If the ROM output word is stored in a clocked buffer register, the next microinstruction can be accessed during the execution of the present one. The present control word is protected by the buffer since the new output word cannot enter the register until the next clock pulse. In this way the necessity of waiting after each microinstruction for the ROM to produce the next control word is avoided.

The major obstacle to this approach is the amount of hardware needed to buffer the ROM outputs. To buffer 120 control lines, a like number of flip flopswill be needed. If discrete flip flops are used this means increasing the amount of hardware by approximately one third. However, there are currently available MSI devices which would reduce the required amount of hardware and could be placed on the same substrates as the ROM's. The use of either counters or other MSI functions capable of acting as clocked output buffers may make this method practical.

Another obstacle to this approach is two clock cycle add time of the present design. In the context of microprogramming, this means that the same location must be accessed continuously for two clock cycles in control words

where additions occur. The control logic needed to accomplish this would probably be a phase flip flop scheme much like the one presently used.

A preliminary design showed that approximately 50 additional gates would be required to produce all the necessary timing conditions. This approach is undesirable both because it requires a substantial amount of control logic external to the memory and because it complicates the timing control. To avoid these problems, it is suggested that a high speed adder be investigated.

4.4.4 Summary of Microprogrammed Control

Microprogrammed control provides a great reduction in the OBP control logic through the replacement of 70 percent of the logic by compact LSI memories. The logic which cannot be programmed in the memory can be effectively partitioned through the use of general purpose logic structures.

Although the speed of presently available P-channel MOS memories presents some design problems, it is anticipated that these will be avoided when large densely packaged CMOS memories become available. The use of CMOS memories will also greatly reduce the power requirements of the OBP. The adoption of a one clock cycle adder will also enhance the advantages offered by this design approach.

Because most of the connections in a microprogrammed control unit are made on the memory chip itself, the number of necessary wired connections is relatively small. Consequently system reliability should be improved.

4.5 Summary of Task 3

Both approaches described above will facilitate the partitioning of the OBP control logic. However, the general purpose logic structure approach is really only a partial solution in that by itself it allows only some of

the reductions that are possible when it is used in conjunction with microprogramming. As mentioned above, it should be possible to mount the control on 52 sixty pin substrates, using the general purpose logic technique alone. If, on the other hand, most of the logic is incorporated into a memory, and general purpose structures are used to partition the rest, it is estimated that 20 sixty pin substrates should be sufficient. If 90 pin packages are made available, the numbers of required substrates for the two techniques can be reduced to 27 and 10 respectively.

Based on these results it appears that the optimum design approach is to employ microprogramming to the maximum extent possible, using general purpose logic structures to partition the remaining logic. Since it has been shown that significant reductions in the required number of substrates can be made by relaxing pin limitations, it is further suggested that the possibility of using substrates with more pins be seriously considered.

It should be noted that although the use of substrates has been assumed throughout this section, the results apply equally to any type of replaceable package (e.g., printed circuit cards, etc.). Substrates were used because they appeared to be the most likely packaging device for the next generation OBP.